What Do You Thing?

1. IOTIGNITE'A GİRİŞ

1.1 IotIgnite Nedir ?

IotIgnite nesnelerin internetindeki nesneyi, internete yani ARCloud'a bağlayan arayüzdür. Bu arayüz sayesinde sensörler, aktivatörler kolayca bağlanıp konfigüre edilebilir.

1.2 IotIgnite Nasıl Çalışır ?

IotIgnite, geliştireceğiniz android uygulamasına kütüphane olarak eklenir. Kimlik doğrulama apisiyle geçerli olan kimlik doğrulaması sonucunda Modiverse uygulamasıyla haberleşerek ARCloud'a bağlanır. Uygulamanızdaki nod ve sensörleri envanter halinde ARCloud tenantınıza yollar ve oradan verileri izlemenize, konfigüre edebilmenize olanak sağlar.

1.3 Hangi Tür Nod ve Sensörler IotIgnite ile Haberleşebilir ?

Üzerinde android işletim sistemi bulunan herhangi bir cihaz ve o cihaza bağlanabilen bütün nod ve sensörler IotIgnite ile haberleşebilir. Örneğin mobil cihazlardaki gömülü sensörler, jiroskop , ivme ölçer, ışık sensörü, manyetik alan sensörü vs.. , mobil cihaza çeşitli bağlantı yollarıyla bağlanmış(bluetooth, wifi , usb serial vs..) Arduino, esp8266 , Raspberry Pi, Spark Photon, pcDuino, Edison, Galileo ve daha buna benzer birçok geliştirme kiti ve bu nodlara bağlı bütün sensörler, android cihaza nod veya sensör olarak tanıtılabilir. IotIgnite bu nod ve sensörleri donanımdan bağımsız olarak soyutlayarak ARCloud'a gönderir. Bunlara ek olarak bağlayacağınız nod veya sensör tamamen sanal da olabilir. Örneğin asal sayı

üreten bir nod tanımlayabilir ve bu noda bağlı sensörler olarak sonu 7 ile biten asal sayı sensörü, sonu 3 ile biten asal sayı sensörü tanımlayabilirsiniz! Neyin nod veya sensör olacağı sizin hayal gücünüze kalmış.

2. IOTIGNITE SDK'SINI KULLANARAK UYGULAMA GELİŞTİRME ORTAMI HAZIRLAMA

Eclipse ile IotIgnite'ı kullanmak için :

I. IotIgniteSDK.zip'i aşağıdaki linkten indirin:
"link buraya"
II. Zipi herhangi bir yere açın. Örneğin workspace'inize olabilir. Açtığınız zipin içerisindeki IotIgnite.jar'ı projenizin "libs/" klasörüne kopyalayın. IotIgnite'ı kullanmaya hazırsınız.

Android Studio ile IotIgnite'ı kullanmak için :

I. IotIgniteSDK.zip'i aşağıdaki linkten indirin:
"link buraya"
II. Zipi herhangi bir yere açın örneğin workspace'iniz olabilir. Zipin içindeki IotIgnite.jar dosyasını yine /libs klasörü altına kopyalayın.
III. build.gradle dosyasına gelince ve dependencies kısmını aşağıdaki gibi düzenleyin :

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'


}
```

IotIgnite'ı kullanmaya hazırsınız!

.aar olarak eklemek içinse :

I. Build.gradle dosyasının repositories ve dependencies kısmı aşağıdaki gibi olmalıdır :

```
repositories {
    mavenCentral()
    flatDir {
        dirs 'libs'
    }
```

```
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile(name:'IoTIgnite/  0.1', ext:'aar')
    testCompile 'junit:junit:4.12'

}
```

3. IOTIGNITE APILERI NELERDİR ?

3.1 IotIgniteManager

IotIgnite SDK'sını kullanmak için giriş noktasıdır. IotIgniteManager instance'ı almak için 3 parametre gereklidir. Bu parametreler :

1) Context : Android Uygulama Contexti.
2) AppKey : Tenant tarafından sağlanan güvenli haberleşme anahtarı.
3) ConnectionCallback : Ignite bağlantı bildirimini belirten listener.

Kullanım örneği;

```
IotIgniteManager mIotIgniteManager = new
IotIgniteManager.Builder().setAppKey('c4d1bf09d50b4e0a8fb158a9bbab845d').setContext(get
ApplicationContext()).setConnectionListener(new ConnectionCallback() {
    @Override
    public void onConnected() {
        Log.i('IgniteSample', 'Ignite Connected!');
    }

    @Override
    public void onDisconnected() {


    }
}).build();
```

IgniteManager bağlanmadan IoTIgnite üzerinde işlem yapılamaz. Ignite'ın

onConnected() callbacki tetiklendikten sonra NodeFactory ile nod oluşturulup kayaşağıdaki gibidir :
dedilebilir. Kayıtlı nodlar silinebilir. IgniteManager'a ait diğer fonksiyonlara javadocta değinilmiştir.

IotIgniteManager aldığı appKey ile kimlik doğrulaması yapar ve geçerli appKey için onConnected callback'ini tetikler. Ignite'ın bağlı olması ARCloud ile haberleşebilir durumda olduğu anlamına gelir.

3.2 Node Nedir, Nasıl Kaydedilir ?

Cihaza bağlı kendi üzerinde sensör bulunduran donanımları gruplamak için oluşturulan sınıftır. Node üzerinde 0..n adet kayıtlı sensör olabilir. Örneğin ArduinoYun, mobil cihaza bağlandığında, ArduinoYun'u nod, ona bağlanan donanımları ise sensör olarak tanımlamak mantıklı olacaktır. Böylece cihaza bağlı farkı donanımlar ayrık şekilde konfigüre edilebilir.

Nod oluşturmak için öncelikle Ignite bağlı durumda olmalıdır. Örnek nod aşağıdaki şekilde oluşturulup kaydedilir :

```
Node myNode = IotIgniteManager.NodeFactory.createNode('myNode', 'myNode',
NodeType.GENERIC);
myNode.register(); // nod basarılı sekilde kaydedildiyse 'true' döner.
```

Her cihaz için **nodeID** eşşiz olmalıdır. Aynı cihazda Ignite'ı kullanan farklı uygulamalar olsa dahi aynı nodeID'yi kullanamaz. Ayrıca "**Built-in Sensors", "Built-in Processors"** nodeID'leri Modiverse uygulaması tarafından kullanıldığından saklı idlerdir.

Noda bağlı bütün işlemler o nodun instance'ı üzerinden yapılır :

- Node kaydetme,silme,kayıt durumu sorgulama, nodun bağlantı durumunu değiştirme, o noda bağlı yeni thing ekleme gibi...
- Kaydedilen nodun ARCloud üzerinde online/offline görünme durumu da yine aynı şekilde *myNode.setConnected(true/false)* fonksiyonuyla gerçekleşir. Node online yapılmadan ona bağlı thingler ile işlem yapılmamalıdır.
- Node apileri javadocta ayrıntılı şekilde anlatılmıştır.

3.3 Thing Nedir, Nasıl Kaydedilir ?

Thing nesneleri IotIgnite'ın en uç birimidir. Gerçek sensörlerden hayali data üreten fonksiyonlara kadar her şey "thing" olarak tanımlanabilir. Her "thing" bir noda bağlı olmak zorundadır. Farklı nodlara bağlı aynı "thing" objeleri olabilir. Yani "thing" nesneleri Nod temelinde eşsiz olmalıdır. Bir thing nesnesi oluşturmak için öncelikle

bağlı bir IotIgnite ve halihazırda kaydedilmiş bir nod gerekir. Kayıtlı nod instance'ı üzerinden thing iki farklı şekilde kaydedilebilir.

- Eğer kaydedilecek thing objesi bir android sensörüyse :

```
Thing myAndroidThing = myNode.createThing(myAndroidSensor, new ThingListener() {
    @Override
    public void onConfigurationReceived(Thing thing) {

    }

    @Override
    public void onActionReceived(String s, String s1, ThingActionData thingActionData) {

    }
});   // Thing basarılı olusturulduysa true döner.

myAndroidThing.register(); // // Thing basarılı olusturulduysa true döner.
```

- Farklı bir "şey" ise :

```
ThingType type = new ThingType('myType','vendor', ThingDataType.STRING);
Thing myOtherThing = myNode.createThing('myUniqueThingID',type,ThingCategory.EXTERNAL,
false, new ThingListener() {
    @Override
    public void onConfigurationReceived(Thing thing) {

    }

    @Override
    public void onActionReceived(String s, String s1, ThingActionData thingActionData) {

    }
});   // Thing basarılı olusturulduysa true döner.

myOtherThing.register();
```

Buradaki onConfigurationReceived ve onActionReceived callbacklerine daha sonra

değineceğiz.

Aynı nodda olduğu gibi thing e bağlı bütün işlemler yine o thing instance'ı üzerinden yapılır. Yine aynı şekilde thing nesnesininde bağlantı durumu *myThing.setConnected(true/false)* fonksiyonu ile belirlenir.

3.4 ThingListener, ThingConfiguration, Actuator Nedir, Nasıl Kullanılır?

Kaydedilen her thing nesnesinin bir listenerı bulunur. Bu listener sayesinde nesne ARCloud üzerinden kontrol edilebilir.

Thing objesi veri üreten bir sensör olacağı gibi olay sonucunda eylemde üretebilir. Örneğin sıcaklık,nem üreten thing objesi sensör görevi görürken, thing olarak tanımlı bir LED'e gönderilen mesajla LED'in kapanıp açılması onun actuator olarak çalıştığını gösterir. Bir thing objesi bu özelliklerden en az birine sahip olacağı gibi hem actuator, hem sensör görevi de görebilir. İşte ThingListener'a ait callbackler bu görevi yapar. Eğer obje sadece veri üreteci olarak tanımlıysa ona inen konfigürasyonda kaç saniyede bir veri göndermesi gerektiği, yolladığı verinin yerel hafızada ne kadar tutulacağı, gönderi eşiği gibi bir çok parametre nesneye ARCloud tarafından iletilir. Yine aynı şekilde obje actuator olarak tanımlıysa sonuçta oluşan eylem onActionReceived() callbackiyle kendisine bildirilir.

Thing nesnelerine ait konfigürasyon ve action mesajları ARCloud tarafında tanımlanır. Konfigürasyon almayan gömülü android sensörler çalışmazken, diğer nesneler için tanımlı varsayılan okuma aralığı 10 dakikadır. Aşağıdaki örnek uygulama konfigürasyon ve actuator kavramlarını daha da pekiştirmenizi sağlayacaktır.

3.5 ThingType, ThingDataType, ThingCategory,ThingActionData Nedir, Nasıl Kullanılır?

Thing nesnesi oluştururken verdiğimiz parametlerden ikisi olan ThingType ThingCategory'e değinelim. ThingType oluşturulan sensör hakkında ARCloud'un fikir sahibi olmasını sağlayacak bildirimi içerir. Sensör ne sensörü, üreticisi kim ne tipte data üretir. Bu bilgileri alır ve thing nesnesi içine kaydeder. Eğer bir android sensörünü thing objesi olarak kullanıyorsanız bu bilgiler zaten sensörün kendisinde bulunduğundan bu parametreleri girmenize gerek kalmaz. Ancak kendinize özgü sensör veya actuator belirlediğiniz durumlarda bu parametreleri girmeniz gerekir. ThingCategory ise eğer bağlanan obje cihazın kendi üzerinde tanımlıysa **ThingCategory.BUILTIN,** bir nod üzerinden geliyorsa **EXTERNAL** olarak tanımlanır. Android sensörler varsayılan olarak **BUILTIN**'dir.

ThingDataType ise oluşturulan sensörün ARCloud'da veya cihazda anlamlı şekilde yorumlanması için yollanan verinin türünü belirtir. ThingDataType şimdilik sadece ***FLOAT,LOCATION,INT ve STRING*** olabilir.

ThingActionData ise olaylar sonucu oluşan eylemlerin gerçekleşmesi için gerekli mesajlaşma yapısını içerir. Bu yapı tamamen ARCloud'ta kuralı yazan kişi ile yazılımı geliştiren kişi arasındadır belli bir standardı yoktur.Örnek olarak LED actuator olarak tanımlıysa ARCloud kuralını yazan kişi ledi açmak için sadece 1-0 mesajı gönderebilir veya mesajı JSON tipinde yollar. Uygulama tarafı ise yollanan mesaj içeriğine göre davranmak zorundadır.

4.IOTIGNITE SDK ÖRNEK PROJE

4.1 DHT11 Sıcaklık&Nem Sensörüyle Değerlerin ARCloud'a İletilmesi ve Nod Üzerindeki Led Kontrolü

Projemizde NodeMcu'yu Dell Gateway'e kablosuz olarak bağlayacağız ve sensör verilerini bu şekilde ileteceğiz.

Proje için gerekli donanım parçaları aşağıdaki gibidir :

1. NodeMCULua
2. DHT11 Sıcaklık&Nem Sensörü
3. 1 Adet LED.
4. Android Gateway(Biz Dell Gateway kullanacağız herhangi bir android cihazda olabilir)

Proje temel olarak iki kısımdan oluşuyor. Birinci kısım gateway üzerinde çalışacak programın yazılması diğeri kısım ise NodeMCULua üzerindeki çalışacak programın yazılması.

NodeMCU Nedir, Niçin Projede Onu Kullanıyoruz ?

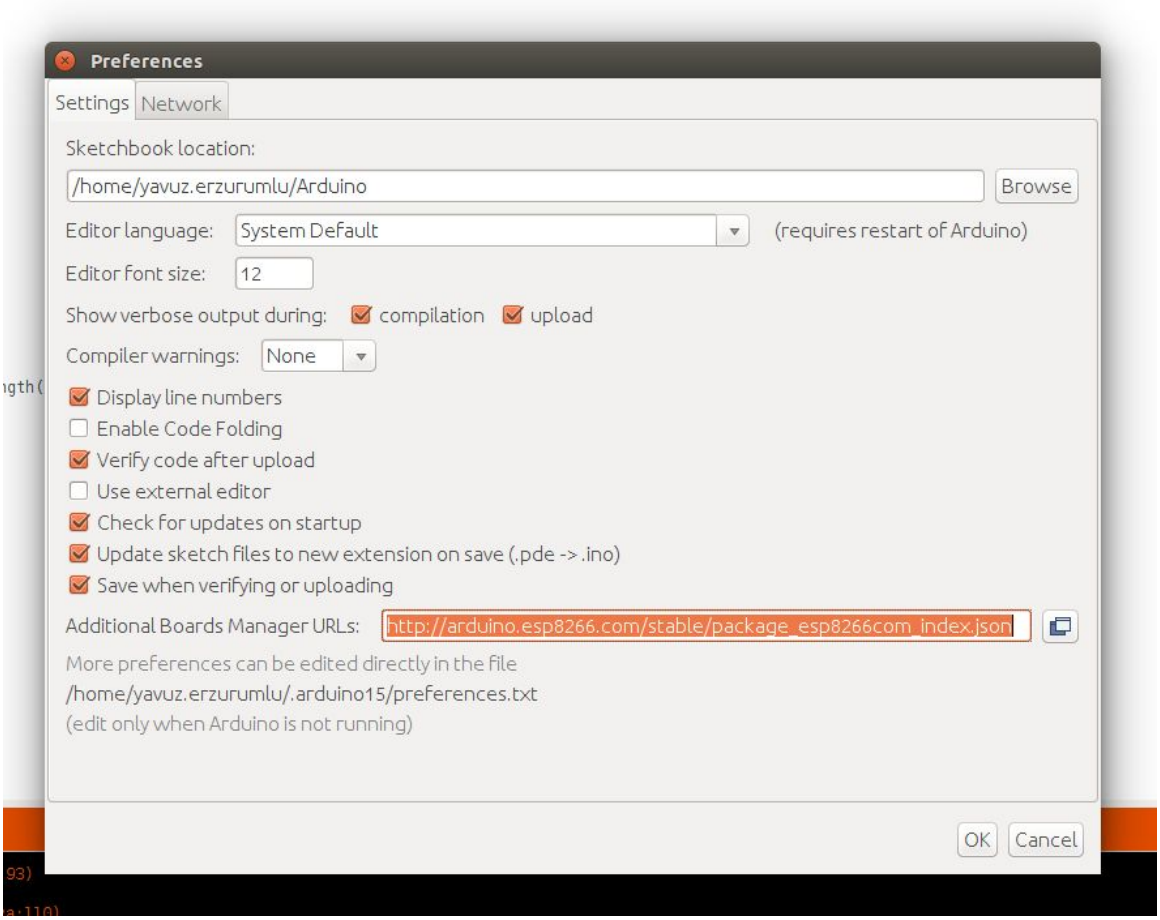
NodeMCU üzerinde Espressif firmasına ait esp8266-12 kablosuz haberleşme modulü bulunduran, usb üzerinden direk programlanabilen uygulama geliştirme entegresidir. Bacak bağlantıları ve donanım aşağıdaki gibidir. NodeMCU üzerindeki uygun fiyatlı kablosuz haberleşme modülü, lua desteklemesi, arduino id ile beraber programlanabilmesi, 10 adet GPIO (Her biri pwm görevi görüyor) pini, 4M dahili hafızası ile fiyat/performans olarak çok yüksek yerlere gelmektedir. Projemizde NodeMCU'yu ArduinoIDE üzerinden programlayacağız.

En son ArduinoIDE'yi aşağıdaki linkten indirin :

- https://www.arduino.cc/en/Main/Software

Kurulumu yaptıktan sonra Dosya/Ayarlar - File/Preferences kısmına gelin ve aşağıdaki ek esp8266 kütühanelerini ekleyin. Böylece arduinoIDE nodeMcu'yu tanıyacak ve programlayabilmenizi sağlayacaktır.

"http://arduino.esp8266.com/stable/package_esp8266com_index.json"



NodeMCU'yu usb ile bilgisayarımıza bağladıktan sonra Tools/Board üzerinden NodeMCU'yu seçelim.

Program temel olarak aşağıdaki işleri yapacaktır :

1. Wifi ağına bağlan.
2. Soket sunucusunu başlat.
3. Kendi ip ve portunu ağ üzerinde yayına başla.
4. Client bağlantısı oluştuğunda sıcaklık ve nem verilerinin gönderileceği sıklığı oku ve o aralıklarla verileri client'a yollamaya başla.
5. Bağlantı koparsa ağa tekrar bağlan, client bağlantısını sonlandır ve yeniden bağlantı bekle.
6. LED eylemi geldiğinde gerçekleştir.

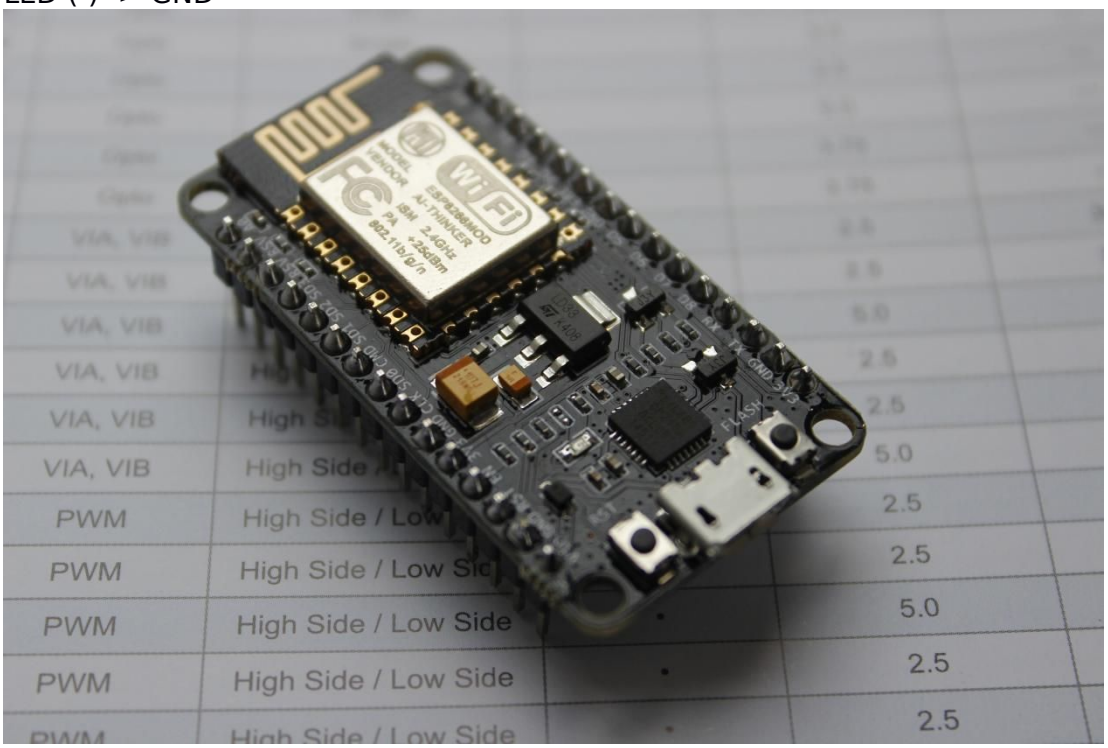
Bu adımları gerçekleştirecek olan uygulama kaynak kodu IgniteSDK.zip içerisinde bulunmaktadır.

Bacak bağlantıları ise aşağıdaki gibi olmalıdır. Vin 3.3v seçilirse sensör düzgün

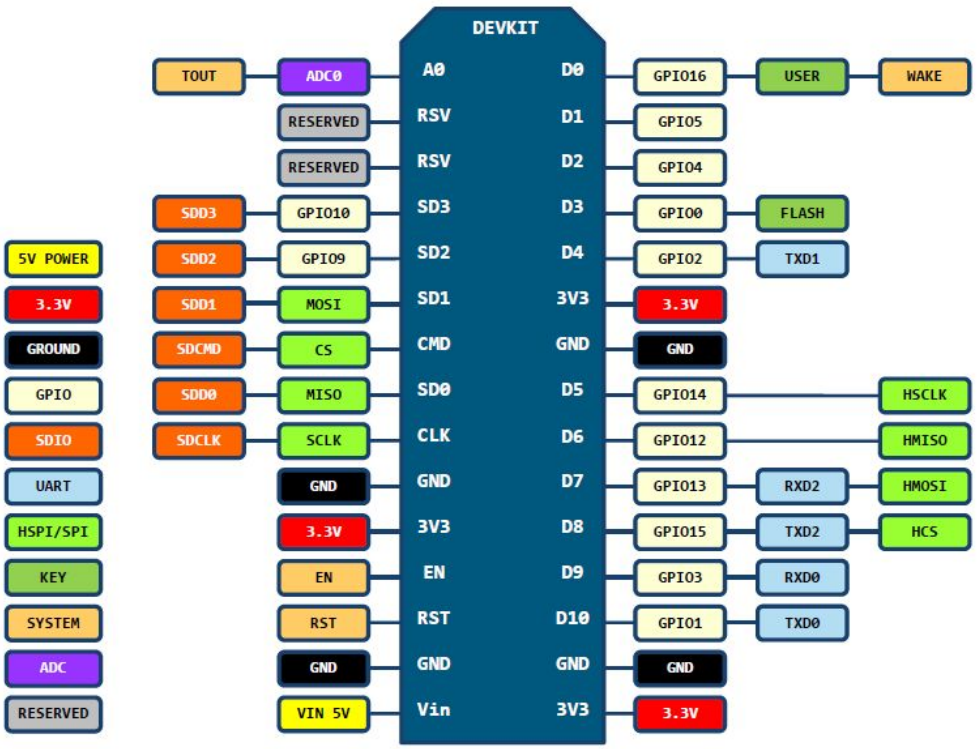
çalışmadığı için 5v’luk çıkışa bağlıyoruz.

DHT11 (+) -> Vin
DHT11(out) -> D4
DHT11(-) -> GND

LED (+) -> D2
LED (-) -> GND



PIN DEFINITION



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Uygulamayı flashlayınca NodeMCU, ATOS ağına bağlanacak 9999 portundan gelecek bağlantı isteğini dinleyecektir. Gateway noda bağlandığında varsayılan olarak 10 saniyede bir verilerini yollayacaktır. Normalde node konfigürasyon almadan data yollamamalıdır. Sadece veri akışını görebilmek için böyle bir ayar giriyoruz.

Şimdi projenin ikinci kısmı olan Android kısmına bakalım. Bu kısımda Gateway'imizi node bağlayacak ve buradan gelen verileri IotIgnite aracılığıyla ARCloud'a yollayacağız.

Android kısmı ise temel olarak aşağıdaki işleri yapmalı :

1. Ağdaki ESP8266'ları tara.
2. IP ve Portu bulunan ESP için Ignite'a bağlan.
3. Ignite bağlantısı başarılı olduysa node ve sensörleri -eğer kayıtlı değilse- Ignite'a kaydet.
4. Sokete bağlan DHT11 verilerini Ignite'ın belirttiği konfigürasyonlarla okumaya başla ve Ignite aracılığıyla yolla.
5. Konfigürasyon ve eylem değişikliklerini dinle.
6. Herhangi durumda IotIgnite veya soket bağlantısı koparsa belirli süre sonunda tekrar bağlanmayı dene.

Eğer uygulamayı Android Studio ile geliştiriyorsanız projenin app/gradle.build'ine gson dependency'sini eklemeniz gerekiyor.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion '23.0.3'

    defaultConfig {
        applicationId 'com.ardic.android.sampleiotigniteproject'
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName '1.0'
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard/android.txt'),
'proguard/rules.pro'
        }
    }
}
repositories {
    mavenCentral()
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat/v7:23.2.1'
    compile 'com.google.code.gson:gson:2.6.2'
}
```

Uygulamayı geliştirirken javadoc görüntülenmiyor. Android Studio'da şu an için doğrudan javadoc desteği yok. Eğer javadocu görüntülemek isterseniz .idea/libraries klasöründeki IoTIgnite_0_1.xml aşağıdaki gibi olmalı :

<root> kısmına javadoc'u açtığınız pathi eklerseniz sorun çözülecektir. Örnek path:

file:///home/yavuz.erzurumlu/DEVELOPMENT/GIT/OFFICIAL-AMP/libs/IoTIgnite/target/apidocs/

```
<component name="libraryTable">
  <library name="IoTIgnite/ 0.1">
    <CLASSES>
      <root url="jar://$PROJECT_DIR$/app/libs/IoTIgnite/ 0.1.jar!/" />
    </CLASSES>
    <JAVADOC>
      <root url="file:// #Path of JavaDoc Source#" />
    </JAVADOC>
    <SOURCES />
  </library>
</component>
```

IotIgnite apileri kimlik doğrulaması olmadan kullanılmadığından uygulamamız önce kimlik doğrulaması yapmalı :

```
mIotIgniteManager = new IotIgniteManager.Builder()
            .setContext(getApplicationContext())
            .setAppKey(TEST_APP_KEY)
            .setConnectionListener(this)
            .build();
```

Doğrulama sonucunda uygulama onConnected() callback'ine düşecek ve nod, thing kayıt işlemlerini buradan gerçekleştireceğiz.

```
@Override
public void onConnected() {
    Log.i(TAG, "Ignite Connected!");
    igniteConnected=true;
    // Ignite Connected Register Node and Things..
    initIgniteVariables();
}
```

Ignite değişkenlerini oluşturup kaydedelim:

```
private void initIgniteVariables(){
    mTempThingType = new ThingType('Temperature Sensor','DHT/  11',
ThingDataType.FLOAT);
    mHumThingType = new ThingType('Humidity Sensor','DHT/  11',ThingDataType.FLOAT);
    mLEDThingType = new ThingType('LED Actuator','LED',ThingDataType.STRING);

    myNode = IotIgniteManager.NodeFactory.createNode(NODE, NODE, NodeType.GENERIC);

    // register node if not registered and set connection.
    if( !myNode.isRegistered() && myNode.register()){
        myNode.setConnected(true,NODE + ' is online');
        Log.i(TAG, myNode.getNodeID() + ' is successfully registered!');
    }else{
        myNode.setConnected(true,NODE + ' is online');
        Log.i(TAG, myNode.getNodeID() + ' is already registered!');
    }
    if(myNode.isRegistered()){

        mTemperatureThing = myNode.createThing(TEMP_THING, mTempThingType,
ThingCategory.EXTERNAL,false,tempThingListener);
        mHumidityThing =
myNode.createThing(HUM_THING,mHumThingType,ThingCategory.EXTERNAL,false,humThing
Listener);
        mLEDThing =
myNode.createThing(LED_THING,mLEDThingType,ThingCategory.EXTERNAL,true,ledThingList
ener);
        registerThingIfNoRegistered(mTemperatureThing);
        registerThingIfNoRegistered(mHumidityThing);
        registerThingIfNoRegistered(mLEDThing);

    }
    // register things...


}
```

Uygulamamız başarılı bir şekilde nod envanterini yolladıktan sonra, nodeMCU'dan gelecek dataları bekleyecektir.

nodeMCU'ya bağlanan soket istemcisini uygulamamızın onCreate()'in de çağıralım.

```
myReaderThread = new ClientThread(DEVICE_IP, DEVICE_PORT, 'read');
myReaderThread.setOnPacketListener(new PacketListener() {
    @Override
    public void onPacketReceived(String packet) {

        if(igniteConnected) {
            onPacketHandler(packet);
            Log.i(TAG, 'Packet Sending... : ' + packet);
        }
    }
});



if (myReaderThread != null) {
    if (BuildConfig.DEBUG) {
        Log.i(TAG, 'Device Ip : ' + DEVICE_IP + ' Device Port : ' + DEVICE_PORT);
    }
    Toast.makeText(getApplicationContext(), 'Device Ip : ' + DEVICE_IP + ' Device Port : ' +
            DEVICE_PORT, Toast.LENGTH_LONG).show();
    myReaderThread.start();
}
else {
    Toast.makeText(getApplicationContext(), 'Connection Error Please Try Again ',
Toast.LENGTH_LONG)
            .show();
    Log.i(TAG, 'myReader Thread is NULL');
}
```

DEVICE_IP, DEVICE_PORT olarak tanımlı adrese bağlantı başarılı olduğunda, sıcaklık nem değerleri onPacketReceived() callback'ine düşecek. Ignite bağlı olduğu sürece bu dataları ARCloud'a yollayacak.

```
private void onPacketHandler(String packet) {
    // handle sensor values
    if (packet.charAt(0) == '#') {
        final String[] values =
                PacketUtils.packetHandler(packet);

        // values[1] /   | temperature

        mTempData = new ThingData();
        mTempData.addData(values[1]);
        mTempData.setDataAccuracy(100);
        mTemperatureThing.sendData(mTempData);

        // values[2] /   | humidity

        mHumData = new ThingData();
        mHumData.addData(values[2]);
        mHumData.setDataAccuracy(100);
        mHumidityThing.sendData(mHumData);

    }

}
```

Varsayılan olarak nodeMcu 10 saniyede bir veri yollamaktadır. Ancak ARCloud'a yeni ayar tanımlanarak bu değerler değiştirilebilir. Üzerinde bulunan LED'e karmaşık olay işleme ile kural tanımlanarak ARCloud'dan kontrol edilebilir.

Benim tanımladığım örnek akışlar ve ayarlar :

1. Sıcaklık > 30 ise LED'i aç, küçükse LED'i kapat.
2. Sıcaklık,Nem verilerinin gönderim sıklığını değiştirip, gözlemleme.

Projenin nodeMCU ve android kaynak kodlarına aşağıdaki adresten ulaşabilirsiniz :

- Kaynak kodlar IgniteSDK.zip içerisindedir.